M&G No. 60001.0270US01/MS303919.1

PROGRAMMABLE OBJECT MODEL FOR NAMESPACE OR SCHEMA LIBRARY SUPPORT IN A SOFTWARE APPLICATION

Related Applications

United States Utility Patent Application by applicant matter number 60001.0263US01/MS303917.1, entitled "Programmable Object Model for Extensible Markup Language Schema Validation," and United States Utility Patent Application by applicant matter number 60001.0264US01/MS303918.1, entitled "Programmable Object Model for Extensible Markup Language Markup in an Application," are hereby incorporated by reference.

Copyright Notice

Field of the Invention

The present invention relates programmable object models. More particularly, the present invention relates to a programmable object model for Namespace or schema library support in a software application.

Background of the Invention

Computer software applications allow users to create a variety of documents to assist them in work, education, and leisure. For example, popular word processing applications allow users to create letters, articles, books, memoranda, and the like. Spreadsheet applications allow users to store, manipulate, print, and display a

variety of alphanumeric data. Such applications have a number of well known strengths including rich editing, formatting, printing, calculation and on-line and off-line editing.

Most computer software applications do not contain all necessary programming for providing functionality required or desired by every potential user. Many programmers often wish to take advantage of an existing application's capabilities in their own programs or to customize the functionality of an application and make it more suitable for a specific set of users or actions. For example, a programmer working in the financial industry may wish to customize a word processor for a user audience consisting of financial analysts editing financial reports. In recent years, the Extensible Markup Language has been used widely as an interchangeable data format for many users. Often users of XML functionality attach or associate one or more XML schema files or XML-based solutions to a document being edited or created by the user. However, users/programmers are limited in their ability to apply XML schema files and other XML-based solutions functionality to a given document because the user/programmer does not have direct and easy access to the Namespace or schema library containing the XML schema files or other XML-based solutions.

Accordingly, there is a need in the art for a programmable object model for allowing a user/programmer to access a Namespace or schema library of XML resources for customizing or otherwise manipulating the resources to enhance the user/programmer's use of XML functionality with a software application document. It is with respect to these and other considerations that the present invention has been made.

Summary of the Invention

The present invention provides methods and systems for allowing a user to programmatically access and utilize a Namespace or schema library containing XML schema files and related XML-based resources for associating those XML-based resources with one or more documents and for customizing the functionality associated with those XML-based resources. Once a user or programmer obtains access to the

Namespace or schema library, the user may programmatically associate XML schema files with XML data in an associated document, and conversely, the user may detect and remove associations of XML schema files with XML data contained in the document. The user may also programmatically associate transformation files with XML data contained in a document and detect and remove existing transformation files associated with XML data contained in the document. The user may also associate other files and executable software associated with XML-based and other document solutions with XML data contained in the document. Additionally, the user may detect and delete the association of XML-based solutions and other types of executable software from association with XML data contained in a document.

These and other features, advantages, and aspects of the present invention may be more clearly understood and appreciated from a review of the following detailed description of the disclosed embodiments and by reference to the appended drawings and claims.

Brief Description of the Drawings

Fig. 1 is a simplified block diagram of a computing system and associated peripherals and network devices that provide an exemplary operating environment for the present invention.

Fig. 2 is a simplified block diagram illustrating interaction between software objects according to an object-oriented programming model.

Fig. 3 is a block diagram illustrating interaction between a document, an attached schema file, and a schema validation functionality model.

Fig. 4 is a block diagram illustrating interaction between a document, a Namespace or schema library and a third party software application.

Detailed Description of the Preferred Embodiment

Embodiments of the present invention are directed to methods and systems for allowing a user to programmatically call a Namespace/Schema library of

XML schema files and XML-based solutions and resources for controlling the association of those files, solutions and resources to one or more documents. These embodiments may be combined, other embodiments may be utilized, and structural changes may be made without departing from the spirit or scope of the present invention. The following detailed description is therefore not to be taken in a limiting senses and the scope of the present invention is defined by the appended claims and their equivalents.

Referring now to the drawings, in which like numerals represent like elements through the several figures, aspects of the present invention and the exemplary operating environment will be described. Fig. 1 and the following discussion are intended to provide a brief, general description of a suitable computing environment in which the invention may be implemented. While the invention will be described in the general context of program modules that execute in conjunction with an application program that runs on an operating system on a personal computer, those skilled in the art will recognize that the invention may also be implemented in combination with other program modules.

Generally, program modules include routines, programs, components, data structures, and other types of structures that perform particular tasks or implement particular abstract data types. Moreover, those skilled in the art will appreciate that the invention may be practiced with other computer system configurations, including hand-held devices, multiprocessor systems, microprocessor-based or programmable consumer electronics, minicomputers, mainframe computers, and the like. The invention may also be practiced in distributed computing environments where tasks are performed by remote processing devices that are linked through a communications network. In a distributed computing environment, program modules may be located in both local and remote memory storage devices.

Turning now to Fig. 1, an illustrative computer architecture for a personal computer 2 for practicing the various embodiments of the invention will be described. The computer architecture shown in Fig. 1 illustrates a conventional personal computer, including a central processing unit 4 ("CPU"), a system memory 6,

including a random access memory 8 ("RAM") and a read-only memory ("ROM") 10, and a system bus 12 that couples the memory to the CPU 4. A basic input/output system containing the basic routines that help to transfer information between elements within the computer, such as during startup, is stored in the ROM 10. The personal computer 2 further includes a mass storage device 14 for storing an operating system 16, application programs, such as the application program 305, and data.

The mass storage device 14 is connected to the CPU 4 through a mass storage controller (not shown) connected to the bus 12. The mass storage device 14 and its associated computer-readable media, provide non-volatile storage for the personal computer 2. Although the description of computer-readable media contained herein refers to a mass storage device, such as a hard disk or CD-ROM drive, it should be appreciated by those skilled in the art that computer-readable media can be any available media that can be accessed by the personal computer 2.

By way of example, and not limitation, computer-readable media may comprise computer storage media and communication media. Computer storage media includes volatile and non-volatile, removable and non-removable media implemented in any method or technology for storage of information such as computer-readable instructions, data structures, program modules or other data. Computer storage media includes, but is not limited to, RAM, ROM, EPROM, EEPROM, flash memory or other solid state memory technology, CD-ROM, DVD, or other optical storage, magnetic cassettes, magnetic tape, magnetic disk storage or other magnetic storage devices, or any other medium which can be used to store the desired information and which can be accessed by the computer.

According to various embodiments of the invention, the personal computer 2 may operate in a networked environment using logical connections to remote computers through a TCP/IP network 18, such as the Internet. The personal computer 2 may connect to the TCP/IP network 18 through a network interface unit 20 connected to the bus 12. It should be appreciated that the network interface unit 20 may also be utilized to connect to other types of networks and remote computer systems. The personal computer 2 may also include an input/output controller 22 for receiving

and processing input from a number of devices, including a keyboard or mouse (not shown). Similarly, an input/output controller 22 may provide output to a display screen, a printer, or other type of output device.

As mentioned briefly above, a number of program modules and data files may be stored in the mass storage device 14 and RAM 8 of the personal computer 2, including an operating system 16 suitable for controlling the operation of a networked personal computer, such as the WINDOWS XP operating system from MICROSOFT CORPORATION of Redmond, Washington. The mass storage device 14 and RAM 8 may also store one or more application programs. In particular, the mass storage device 14 and RAM 8 may store an application program 305 for creating and editing an electronic document 310. For instance, the application program 305 may comprise a word processing application program, a spreadsheet application, a contact application, and the like. Application programs for creating and editing other types of electronic documents may also be used with the various embodiments of the present invention. A schema file 330and a namespace/schema library 400, described below, are also shown.

Exemplary embodiments of the present invention are implemented by communications between different software objects in an object-oriented programming environment. For purposes of the following description of embodiments of the present invention, it is useful to briefly to describe components of an object-oriented programming environment. Fig. 2 is a simplified block diagram illustrating interaction between software objects according to an object-oriented programming model. According to an object-oriented programming environment, a first object 210 may include software code, executable methods, properties, and parameters. Similarly, a second object 220 may also include software code, executable methods, properties, and parameters.

A first object 210 may communicate with a second object 220 to obtain information or functionality from the second object 220 by calling the second object 220 via a message call 230. As is well know to those skilled in the art of object-oriented programming environment, the first object 210 may communicate with the second object 220 via application programming interfaces (API) that allow two

disparate software objects 210, 220 to communicate with each other in order to obtain information and functionality from each other. For example, if the first object 210 requires the functionality provided by a method contained in the second object 220, the first object 210 may pass a message call 230 to the second object 220 in which the first object identifies the required method and in which the first object passes any required parameters to the second object required by the second object for operating the identified method. Once the second object 220 receives the call from the first object, the second object executes the called method based on the provided parameters and sends a return message 250 containing a value obtained from the executed method back to the first object 210.

For example, in terms of embodiments of the present invention, and as will be described below, a first object 210 may be a third party customized application that passes a message to a second object such as an Extensible Markup Language schema validation object whereby the first object identifies a method requiring the validation of a specified XML element in a document where the specified XML element is a parameter passed by the first object with the identified method. Upon receipt of the call from the first object, according to this example, the schema validation object executes the identified method on the specified XML element and returns a message to the first object in the form of a result or value associated with the validated XML element. Operation of object-oriented programming environments, as briefly described above, are well known to those skilled in the art.

As described below, embodiments of the present invention are implemented through the interaction of software objects in the use, customization, and application of components of the Extensible Markup Language (XML). Fig. 3 is a block diagram illustrating interaction between a document, an attached schema file, and a schema validation functionality module. As is well known to those skilled in the art, the Extensible Markup Language (XML) provides a method of describing text and data in a document by allowing a user to create tag names that are applied to text or data in a document that in turn define the text or data to which associated tags are applied. For example referring to Fig. 3, the document 310 created with the application 305 contains

text that has been marked up with XML tags 315, 320, 325. For example, the text "Greetings" is annotated with the XML tag <title>. The text "My name is Sarah" is annotated with the <body> tag. According to XML, the creator of the <title> and <body> tags is free to create her own tags for describing the tags to which those tags will be applied. Then, so long as any downstream consuming application or computing machine is provided instructions as to the definition of the tags applied to the text, that application or computing machine may utilize the data in accordance with the tags. For example, if a downstream application has been programmed to extract text defined as titles of articles or publications processed by that application, the application may parse the document 310 and extract the text "Greetings," as illustrated in Fig. 3 because that text is annotated with the tag <title>. The creator of the particular XML tag naming for the document 310, illustrated in Fig. 3, provides useful description for text or data contained in the document 310 that may be utilized by third parties so long as those third parties are provided with the definitions associated with tags applied to the text or data.

According to embodiments of the present invention, the text and XML markup entered into the document 310 may be saved according to a variety of different file formats and according to the native programming language of the application 305 with which the document 310 is created. For example, the text and XML markup may be saved according to a word processing application, a spreadsheet application, and the like. Alternatively, the text and XML markup entered into the document 310 may be saved as an XML format whereby the text or data, any applied XML markup, and any formatting such as font, style, paragraph structure, etc. may be saved as an XML representation. Accordingly, downstream or third party applications capable of understanding data saved as XML may open and consume the text or data thus saved as an XML representation. For a detailed discussion of saving text and XML markup and associated formatting and other attributes of a document 310 as XML, see U.S. Patent Application entitled "Word Processing Document Stored in a Single XML File that may be Manipulated by Applications that Understanding XML," U.S. Serial No. 10/187,060, filed June 28, 2002, which is incorporated herein by reference as if fully set out herein.

In order to provide a definitional framework for XML markup elements (tags) applied to text or data, as illustrated in Fig. 3, XML schema files are created which contain information necessary for allowing users and consumers of marked up and stored data to understand the XML tagging definitions designed by the creator of the document. Each schema file also referred to in the art as a XSD file preferably includes a listing of all XML elements (tags) that may be applied to a document according to a given schema file. For example, a schema file 330, illustrated in Fig. 3, may be a schema file containing definitions of certain XML elements that may be applied to a document 310 including attributes of XML elements or limitations and/or rules associated with text or data that may be annotated with XML elements according to the schema file. For example, referring to the schema file 330 illustrated in Fig. 3, the schema file is identified by the Namespace "intro" the schema file includes a root element of <introCard>.

According to the schema file 330, the <introCard> element serves as a root element for the schema file and also as a parent element to two child elements <title> and <body>. As is well known to those skilled in the art, a number of parent elements may be defined under a single root element, and a number of child elements may be defined under each parent element. Typically, however, a given schema file 330 contains only one root element. Referring still to Fig. 3, the schema file 330 also contains attributes 340 and 345 to the <title> and <body> elements, respectfully. The attributes 340 and 345 may provide further definition or rules associated with applying the respective elements to text or data in the document 310. For example, the attribute 345 defines that text annotated with the <title> element must be less than or equal to twenty-five characters in length. Accordingly, if text exceeding twenty-five characters in length is annotated with the <title> element or tag, the attempted annotation of that text will be invalid according to the definitions contained in the schema file 330.

By applying such definitions or rules as attributes to XML elements, the creator of the schema may dictate the structure of data contained in a document associated with a given schema file. For example, if the creator of a schema file 330 for defining XML markup applied to a resume document desires that the experience section

of the resume document contain no more than four present or previous job entries, the creator of the schema file 330 may define an attribute of an <experience> element, for example, to allow that no more than four present or past job entries may be entered between the <experience> tags in order for the experience text to be valid according to the schema file 330. As is well known to those skilled in the art, the schema file 330 may be attached to or otherwise associated with a given document 310 for application of allowable XML markup defined in the attached schema file to the document 310. According to one embodiment, the document 310 marked up with XML elements of the attached or associated schema file 330 may point to the attached or associated schema file by pointing to a uniform resource identifier (URI) associated with a Namespace identifying the attached or associated schema file 330.

According to embodiments of the present invention, a document 310 may have a plurality of attached schema files. That is, a creator of the document 310 may associate or attach more than one schema file 330 to the document 310 in order to provide a framework for the annotation of XML markup from more than one schema file. For example, a document 310 may contain text or data associated with financial data. A creator of the document 310 may wish to associate XML schema files 330 containing XML markup and definitions associated with multiple financial institutions. Accordingly, the creator of the document 310 may associate an XML schema file 330 from one or more financial institutions with the document 310. Likewise, a given XML schema file 330 may be associated with a particular document structure such as a template for placing financial data into a desirable format.

According to embodiments of the present invention, a collection of XML schema files and associated document solutions may be maintained in a Namespace or schema library located separately from the document 310. The document 310 may in turn contain pointers to URIs in the Namespace or schema library associated with the one or more schema files attached to otherwise associated with the document 310. As the document 310 requires information from one or more associated schema files, the document 310 points to the Namespace or schema library to obtain the required schema definitions. For a detailed description of the use of an operation of Namespace or

schema libraries, see U.S. Patent Application entitled "System and Method for Providing Namespace Related Information," U.S. Serial No. 10/184,190, filed June 27, 2002, and U.S. Patent Application entitled "System and Method for Obtaining and Using Namespace Related Information for Opening XML Documents," U.S. Serial No. 10/185,940, filed June 27, 2002, both U.S. patent applications of which are incorporated herein by reference as if fully set out herein. For a detailed description of a mechanism for downloading software components such as XML schema files and associated solutions from a Namespace or schema library, see US Patent Application entitled Mechanism for Downloading Software Components from a Remote Source for Use by a Local Software Application, US Serial No. 10/164,260, filed June 5, 2002.

Referring still to Fig. 3, a schema validation functionality module 350 is illustrated for validating XML markup applied to a document 310 against an XML schema file 330 attached to or otherwise associated with the document 310, as described above. As described above, the schema file 330 sets out acceptable XML elements and associated attributes and defines rules for the valid annotation of the document 310 with XML markup from an associated schema file 330. For example, as shown in the schema file 330, two child elements <title> and <body> are defined under the root or parent element <introCard>. Attributes 340, 345 defining the acceptable string length of text associated with the child elements <title> and <body> are also illustrated. As described above, if a user attempts to annotate the document 310 with XML markup from a schema file 330 attached to or associated with the document in violation of the XML markup definitions contained in the schema file 330, an invalidity or error state will be presented. For example, if the user attempts to enter a title string exceeding twenty-five characters, that text entry will violate the maximum character length attribute of the <title> element of the schema file 330. In order to validate XML markup applied to a document 310, against an associated schema file 330, a schema validation module 350 is utilized. As should be understood by those skilled in the art, the schema validation module 350 is a software module including computer executable instructions sufficient for comparing XML markup and associated text entered in to a

document 310 against an associated or attached XML schema file 330 as the XML markup and associated text is entered in to the document 310.

According to embodiments of the present invention, the schema validation module 350 compares each XML markup element and associated text or data applied to the document 310 against the attached or associated schema file 330 to determine whether each element and associated text or data complies with the rules and definitions set out by the attached schema file 330. For example, if a user attempts to enter a character string exceeding twenty-five characters annotated by the <title> elements 320, the schema validation module will compare that text string against the text string attribute 340 of the attached schema file 330 and determine that the text string entered by the user exceeds the maximum allowable text string length. Accordingly, an error message or dialogue will be presented to the user to alert the user that the text string being entered by the user exceeds the maximum allowable character length according to the attached schema file 330. Likewise, if the user attempts to add an XML markup element between the <title> and the <body> elements, the schema validation module 350 will determine that the XML markup element applied by the user is not a valid element allowed between the <title> and <body> elements according to the attached schema file 330. Accordingly, the schema validation module 350 will generate an error message or dialogue to the user to alert the user of the invalid XML markup.

Programmable Object Model for Namespaces/Schema Libraries

As described above with reference to Fig. 3, in order to provide a definitional and rules-oriented framework for applying Extensible Markup Language (XML) markup to a document 310, one or more schema files 330 may be associated or attached to the document for setting definitions and rules governing the application of XML markup elements corresponding to a given schema file 330 to a document 310. As described, a plurality of XML schema files and other document solutions, for example pre-structured templates, may be attached to or associated with a single XML document 310. Moreover, as described above, a number of different XML schema files

identified by a Namespace identification and a number of document solutions may be stored in a Namespace or schema library apart from the document 310. According to embodiments of the present invention, users are allowed to programmatically call the Namespace or schema library associated with one or more documents 310 for customizing or otherwise manipulating schema file Namespaces and associated definitions, rules, resources, and solutions associated with various Namespace identifiers contained in the Namespace or schema library.

Fig. 4 is a block diagram illustrating interaction between a document 310, a Namespace or schema library 400 and a third party application 450. According to embodiments of the present invention, users may programmatically call the Namespace library 400 via a set of object-oriented message calls or application programming interfaces 470 for modifying the contents or operation of individual schema files 410, 430 or resources 420, 440 associated with schema files identified in the Namespace library 400. The user may communicate with the Namespace library from the application 305 or from a third party program 450 via a set of object-oriented message calls, and the third party program may be developed using a variety of programming languages, such as C, C++, C#, Visual Basic, and the like.

By having access to the Namespace library through a set of application programming interfaces 470, the user may programmatically associate one or more additional XML schema files or Namespaces with XML data, and conversely, the user may detect and remove existing associations between one or more XML schema files and XML data or markup applied to the document 310. The user may also programmatically associate Extensible Stylesheet Language Transformation (XSLT) with XML data applied to a document, and conversely, the user may detect and remove existing XSLT transforms from association with XML data applied to the document 310. Moreover, the user may programmatically associate other files and executable software applications with XML data applied to the document 310 and detect and remove existing associations of other software applications and files with XML data.

For example, the Namespace 430 illustrated in the Namespace library 400 may contain a solution comprised of a pre-formatted structure for a resume

document template. When that solution is applied to the document 310, associated schema definitions and rules designed by the creator of the resume template document will be applied to XML markup and associated text entered into the document 310. If a schema file associated with a resume document template requires that an experience section of a resume document must have at least three past or present job descriptions, that schema definition will be applied to the document 310 such that at least three job descriptions must be entered by a subsequent user in the experience section in order for the XML document 310 to be validated by a schema validation module 350. Continuing with this example, if such a resume document template schema file is associated with the document 310, and a user desires to remove the association of that schema file with the document 310, the user may do so programmatically from a third party program by sending an object-oriented message call to the Namespace library 450 or to the application 305 with a provided application programming interface for directing the removal of the association of the resume document template schema file from the document 310.

The following is a description of objects and associated properties comprising object-oriented message calls or application programming interfaces that allow a user to programmatically access the Namespace library 400 as described above. Following each of the objects and associated properties set out below is a description of the operation and functionality of the object or associated property.

Application object

The following are methods and properties of the object.

.XMLNamespaces property

A read only pointer to an XMLNamespaces collection which represents the Namespace library available to the application.

XMLNamespaces collection object - an object providing access to the XMLNamespace objects. It represents the Namespace library. Each XMLNamespace object in the collection represents a single and unique Namespace in the Namespace library. The following are methods and properties of the object.

.Add() method

A method creating and adding to the collection a new XMLNamespace object. It is used to register a new Namespace in the Namespace library. It returns a new XMLNamespace object. It can accept the following parameters.

Path – pointer to the schema file for the Namespace. The pointer can be a file path represented as a string.

NamespaceURI – the URI of the Namespace that represents the schema. The URI can be a text string.

Alias – a text string representing an alternate (more user-friendly) name for the Namespace that the programmer may specify.

InstallForAllUsers – a flag indicating whether the new Namespace in the Namespace library should be available to all users of the computer or only the current user.

.Application property

A read only pointer to the application object representing the application of this object model.

.Count property

A read only property returning the number of registered Namespaces in the Namespace library. The property is the same as the total number of XMLNamespace objects in the XMLNamespaces collection.

.Creator property

A read only pointer to the creator of the object.

.InstallManifest() method

A method for installing solution manifests that register Namespaces in the Namespace library. It can accept the following parameters.

Path – pointer to the manifest file for the manifest. The pointer can be a file path represented by a text string.

InstallForAllUsers - a flag indicating whether the new Namespaces installed in the Namespace library by the manifest should be available to all users of the computer or only the current user.

.Item() method

A method for accessing the individual members of this collection using an numerical index or a search keyword. The method can accept the following parameters.

Index – a number representing the position of the requested XML Namespace object in the Namespace library. The index can also be a text string representing the alias or the URI of the requested Namespace.

.Parent property

A read only property returning the parent object of the collection. This property returns a pointer to the application from which the XMLNamespaces collection is accessed.

XMLNamespace object – an object representing an individual Namespace entry in the Namespace library (and an individual item in the XMLNamespaces collection). The following are methods and properties of the object.

.Alias property

A property for controlling the alias the programmer associates with the Namespace. It can support the following parameter.

AllUsers – a flag indicating whether the alias is available to all users or just the current user.

.Application property

A read only pointer to the Application object representing the application of this object model.

.AttachToDocument() method

A method for attaching the schema of the Namespace represented by the object to the selected document. It supports the following parameters:

Document – a pointer to the document to which the schema is requested to be attached.

.Creator property

A read only pointer to the creator of the object.

.DefaultTransform property

A property that points to the default XSLT transformation associated with this Namespace. It can support the following parameter.

AllUsers – a flag indicating whether the default transformation setting should affect all users of the machine or only the current user.

.Delete() method

A method for removing the XMLNamespace object from the collection and destroying it, effectively removing the Namespace association represented by this object from the Namespace library.

.Location property

A read only property that controls the location of the schema associated with the Namespace represented by the XMLNamespace object. It can support the following parameter.

AllUsers – a flag indicating whether the schema location setting should affect all users of the machine or only the current user.

.Parent property

A read only property returning the parent object of the XMLNamespace object. This property returns a pointer to the XMLNamespaces collection of which the object is a member.

.URI property

A read only property returning the URI of the Namespace represented by the object.

.XSLTransforms property

A read only pointer to the XSLTransforms collection representing XSLT transformations associated with the Namespace represented by the object.

XSLTransforms object - an object providing access to the XSLTransforms objects each of which represents a single and unique XSLT transform associated with a Namespace in the Namespace library. The following are methods and properties of the object.

.Add() method

A method for creating and adding to the collection a new XSLTransform object. It is used to associate a new XSLT transformation with a Namespace in the Namespace library. It returns a new XSLTransform object. It can accept the following parameters.

Location – pointer to the XSLT file; can be a file path represented as a text string.

Alias – a text string representing an alternate (more user-friendly) name for the XSLT transformation that the programmer may specify.

InstallForAllUsers – a flag indicating whether the new Namespace in the Namespace library should be available to all users of the computer or only the current user.

.Application property

A read only pointer to the Application object representing the application of this object model.

.Count property

A read only property returning the number of registered XSLT transforms for a given Namespace in the Namespace library. It is the same as the total number of XSLTransform objects in the XSLTransforms collection.

.Creator property

A read only pointer to the creator of the object.

.Item() method

A method for accessing the individual members of this collection using an numerical index or a search keyword. It can accept the following parameters.

Index – a number representing the position of the requested XSLTransform object in the Namespace library. The index can also be a text string representing the alias of the requested XSL transform.

.Parent property

A read only property returning the parent object of the collection. This property returns a pointer to the application from which the XSLTransforms collection is accessed.

XSLTransform object - an object representing an XSLT transformation associated with a Namespace in the Namespace library. The following are methods and properties of the object.

.Alias property

A property for controlling the alias the programmer associated with the XSLT transform in the Namespace library. It can support the following parameter.

AllUsers – a flag indicating whether the alias is available to all users or just the current user.

.Application property

A read only pointer to the Application object representing the application of this object model.

.Creator property

A read only pointer to the creator of the object.

.Delete() method

A method for removing the XSLTransform object from the collection and destroying it, effectively removing the association between the XSLT transform and its Namespace in the Namespace library.

.Location property

A read only property that controls the location of the XSLT transform associated with the given Namespace and represented by the XSLTransform object. It can support the following parameter.

AllUsers – a flag indicating whether the XSLT transform location setting should affect all users of the machine or only the current user.

.Parent property

A read only property returning the parent object of the XSLTransform object. This property returns a pointer to the XSLTransforms collection of which the object is a member.

As described herein, methods and system are provided for allowing a user to programmatically call the resources identified in an Extensible Markup Language Namespace or schema library for customizing or otherwise modifying the association of resources identified or contained in the Namespace or schema library with one or more associated documents. It will be apparent to those skilled in the art

that various modifications or variations may be made in the present invention without departing from the scope or spirit of the invention. Other embodiments of the invention will be apparent to those skilled in the art from consideration of the specification and practice of the invention disclosed herein.

5